UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

Trabajo fin de grado

Connectivity Automation for 5G Networks

Alejandro Alcalá Álvarez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Connectivity Automation for 5G Networks

Autor: Alejandro Alcalá Álvarez
Tutor: Víctor López Álvarez

mayo 2021

Alejandro Alcalá Álvarez
Connectivity Automation for 5G Networks

Alejandro Alcalá Álvarez
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Prefacio

Alejandro Alcalá Álvarez

Agradecimientos

En primer lugar, quería agradecer a mi familia por todo lo que ha hecho por mi y por todo el apoyo que me ha dado durante todos estos años en la universidad y sobretodo verlos felices con todos los progresos que estaba haciendo en ella.

También quería acordarme de todas esas personas de mi familia que ya no están, pero se que estarán orgullosos de todo lo que he conseguido.

En segundo lugar, quería agradecer a todos mis amigos por esos momentos de desconexión para poder volver más fuerte y por toda esa felicidad que me dais en cualquier momento.

Por último, quería agradecer a mi tutor Víctor por la oportunidad que me ha dado para realizar este trabajo de fin de grado y por todo el esfuerzo, trabajo y el apoyo que me ha dado durante la realización del trabajo.

Solo puedo deciros gracias a todos por todos los momentos vividos con vosotros y que no me falten nunca.

Resumen

Este trabajo fin de grado consta de una investigación sobre la quinta generación de comunicaciones, dónde esta generación tiene como objetivo **mejorar los servicios en tres clases:** una de estas clases son las comunicaciones de ultra-confianza a baja latencia (URLLC), las comunicaciones masivas entre máquinas (mMTC) y las comunicaciones con banda ancha de móvil mejorada (eMBB). Estos **servicios van a permitir** su uso en industrias y va a tener un gran impacto en la sociedad. El trabajo en el 5G empezó en el año 2010 y ha tenido tres fases de desarrollo, dónde estas fases han tenido diferentes objetivos que han sido marcados por el 3GPP. La primera fase del desarrollo de esta nueva tecnología es el desarrollo teórico de la generación. La segunda fase del desarrollo es la puesta en práctica los desarrollos teóricos de la anterior fase. Por último, la última fase es probar todos los conceptos que han sido aprobados en la anterior fase, para poder realizar prácticas en la sociedad actual y comercializarlo. Todos los objetivos que tiene marcado esta nueva generación de comunicaciones se pueden dar por la división de slices en la red. Una slice es un corte en la red que puede tener diferente funcionamiento al que tiene la red a la que pertenece. Estas slices pueden ser de dos tipos: **soft slicing** y **hard slicing**. El **soft slicing** consiste en una división de la red que depende de los QoS que necesita la aplicación y, cada slice se configura a partir de los QoS. El **hard slicing** se basa en la capacidad que tiene el 5G para replicar y virtualizar los nodos que hay dentro de la red.Para este trabajo fin de grado se ha realizado un proyecto de creación de un **Network Slice Controller** que pueda recibir configuraciones de slices del tipo soft y así poder configurar nuestra red dependiendo de las necesidades que se tenga. Este proyecto se ha diseñado en diferentes fases, dónde se han utilizado dos bases de datos no relaciones para poder almacenar las diferentes configuraciones, estas configuraciones se envían a través de una North-Bound Interface creada a partir de las especificaciones del IETF. Además, se ha desarrollado un Network Slice Orchestrator, que se encarga de obtener los diferentes parámetros de la configuración de la red y, esté comprueba que la configuración esté bien formada y la envía a un coordinador de red para que este coordinador pueda configurar la red con nuestras necesidades.

Palabras clave

5G, Network Slicing, Programacion de redes, trabajo fin de grado

Abstract

This Bachelor Thesis consists of an investigation on the fifth generation of communications, where this generation aims to **improve services in three classes:** enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency Communications (URLLC) and massive Machine Type Communications (mMTC). These **services will allow** their use in industries and will have a great impact on society. Work on 5G began in 2010 and has had three development phases, where these phases have had different objectives that have been set by 3GPP. The first phase of the development of this new technology is the theoretical development of the generation. The second phase of development is the implementation of the theoretical developments of the previous phase. Finally, the last phase is to test all the concepts that have been approved in the previous phase, to be able to carry out practices in today's society and market it. All the objectives set by this new generation of communications can be given by the division of slices in the network. A slice is a cut in the network that may function differently than the network to which it belongs. These slices can be of two types: **soft slice** and **hard slice**. The **soft slicing** consists of a network division that depends on the QoS we need for our application and each slice is configured from the QoS. The **hard slicing** is based on the ability of 5G to replicate and virtualize the nodes within the network. After the research on the fifth generation of communications, a project has been carried out to create a **Network Slice Controller** that can receive soft-type slices configurations and thus be able to configure our network according to the needs it has. This project has been designed in different phases, where two non-related databases have been used to store the different configurations, these configurations are sent through an North-Bound Interface created from the IETF specifications. In addition, a Network Slice Orchestrator has been developed, which is responsible for obtaining the different parameters of the network configuration and verifies that the configuration is well formed and sends it to a network coordinator so that this coordinator can configure the network with our needs.

Keywords

5G, Network Slicing, network programming, final degree project

Índice

Listas

Lista de códigos

Lista de figuras

1

Introducción

La quinta generación de las comunicaciones inalámbricas y de móviles ya esta aquí. Esta quinta generación va a tener un impacto muy grande en la sociedad y en cualquier industria, no solo en las TIC (Tecnologías de la Información y la Comunicación). El 5G va a permitir tener una velocidad de experiencia muy alta en cualquier momento y dónde cualquier persona quiera. Esto se debe a que el 5G aumenta tiene un gran aumento de la velocidad máxima. Esta nueva tecnología nos va a llevar a una nueva experiencia de conectividad entre los seres humanos. El 5G se va a utilizar y se esta utilizando ahora mismo en la realidad aumentada y virtual, en el video free-viewpoint y en la telepresencia. El 5G se basa en estos tres pilares:

- Las comunicaciones de ultra-confianza a baja latencia (URLLC).
- Las comunicaciones masivas entre máquinas (mMTC).
- Una banda ancha de móvil mejorada (eMBB).

Una de las claves por la que el 5G va a provocar este impacto en la sociedad y en las industrias es debido a su clave principal que es el Netowrk Slicing. El network slicing será la clave para dar la flexibilidad necesaria para su funcionamiento a las redes 5G. A grandes rasgos y más adelante se entrará en detalle sobre las network slices, que son redes personalizadas e independientes dentro de una misma infraestructura de red, dónde cada una de estas slices se pueden controlar y gestionar.

1.1. Motivación

La quinta generación de comunicaciones móviles e inalámbricas va a tener un gran impacto tanto en la sociedad y en las industrias, no solo en las industrias de la información y las TIC. El 5G va a permitir comunicaciones ultra fiables de baja latencia y comunicaciones masivas de tipo máquina. La motivación para hacer esta nueva generación de comunicaciones es fomentar la cuarta revolución industrial. Esta tecnología va a desempeñar una función muy importante para el sector del automóvil y del transporte porque va a permitir dar un salto en las formas de la conducción, por ejemplo en la conducción autónoma de los coches. Otra aspecto en el que se puede dar un salto en el sector del automóvil es en la protección de los usuarios del automóvil y de los peatones. También, esta quinta

generación de comunicaciones va a ayudar a mejorar en la eficiencia y sobretodo en la seguridad del transporte ferroviario. Como las comunicaciones van a tener baja latencia y van a ser ultra confiables permitirá controlar maquinas en zonas de difícil acceso, sobretodo en los campos de la minería o de la construcción. Por otro lado, esta generación viene motivada para revolucionar en el campo de los servicios de la salud con la posibilidad de utilizar herramientas quirúrgicas de forma inalámbrica. Por último, viene motivada para acelerar el crecimiento de las ciudades a las SmartCities para dar un salto de calidad en la forma de vida en las ciudades.

1.2. Organización de la memoria

A continuación, se explicará como se ha organizado la memoria de este TFG. En el sección 2.1.1, se va a hacer una breve introducción al 5G, dónde en la sección 2.1.2 se adentrará en explicar como ha sido el desarrollo que han realizado las diferentes empresas para llevar a cabo el 5G y las diferentes fases que ha tenido y, por último, en la sección 2.1.3 se va hablar de las distintas versiones que ha tenido el 5G en el tiempo hasta llegar a la versión LTE.

La sección 2.2 se centrará en el Network Slice y el significado que tiene la palabra slice dentro de una red.En la sección 2.2.1 se hablará del funcionamiento de las slices de la red dentro de la red y ver las diferencias entre esta nueva tecnología con las versiones anteriores que han estado implantadas en la sociedad.Por último, en la sección 2.2.2 se explicará los diferentes tipos de slice y como se pueden configurar para tener el funcionamiento esperado dentro de la red.

En el capítulo 3 se va hablar del diseño seguido en el proyecto que se ha realizado. Este capítulo esta divido en tres secciones. La sección 3.1 realiza un pequeño resumen sobre el modelo que se ha utilizado para crear el diseño del Network Slice Controller North-Bound Interface. En la sección 3.2 habrá un breve resumen de la RFC utilizada en el proyecto y de como esta RFC se ha introducido en el proyecto. Por último, en la sección 3.3 habrá un resumen del modelo utilizado para este proyecto y de como se ha integrado ese modelo en el proyecto.

En el capítulo 4 se va a describir las diferentes fases de desarrollo que ha tenido el proyecto. En la sección 4.1 se van a describir los pasos utilizados para la creación del Network Slice Controller North-Bound Interface, dónde se han especificado los comandos que se han utilizado y se va a poder visualizar partes del código realizado en esta primera fase . En la sección 4.2 se va a poder visualizar y comprender el código para la creación de las bases de datos del proyecto y como se ha introducido con la anterior fase del diseño. En la sección 4.3 se va a describir todos el desarrollo que ha tenido la creación del Network Slice orchestrator. Por último, en la sección 4.4 se han descrito todos los scripts que se han utilizado para poder replicar en real todas las configuraciones de red que llegaban desde el Network Slice Controller North-Bound.

Por último, se ha incluido el capítulo 5, que es el capítulo referentes a las pruebas realizadas. En

este capítulo se van a describir todas las pruebas que se han realizado en las diferentes fases de desarrollo del proyecto

2

Estado del arte

2.1. 5G

2.1.1. Introducción al 5G

La quinta generación de comunicaciones inalámbricas y de móviles van a tener un gran impacto en la sociedad y en la industria, mucho más allá del campo de las ICT. La quinta generación va a tener un gran aumento en la velocidad máxima y va a permitir una alta velocidad de experiencia al usuario en cualquier parte del mundo y en cualquier instante. Otro aspecto, es que va a mejorar el ancho de banda en los dispositivos móviles. El 5G [1] va a llevar al siguiente nivel de la conectividad entre los seres humanos. Algunos ejemplos dónde se va a utilizar esta nueva tecnología son la realidad aumenta y virtual y los vídeos free-viewpoint, por último, va dar un gran salto en la conducción autónoma en el sector del automóvil. El 5G es una nueva generación de comunicaciones inalámbricas y móviles, la cuál va a mejorar a la actual generación de comunicaciones que es el 4G.

Los principales fundamentos sobre la que se construye el 5G son los siguientes:

- Comunicaciones ultra-confiables de baja latencia (URLLC).
- Comunicaciones masivas entre maquinas (eMTC).
- Banda de ancha mejorada (eMBB).

2.1.2. Desarrollo del 5G

Las primeras investigaciones de la quinta generación de las comunicaciones empezaron en el año 2010 y continúan en la actualidad para poder seguir evolucionando esta nueva generación [2]. Estas investigaciones fueron desarrolladas conjuntamente entre la Comisión Europea, todos los fabricantes de las tecnologías de la información y las comunicaciones, los operadores de telecomunicaciones, los proveedores de servicio, las PYMES y las instituciones de investigación. Todos los desarrolladores anteriores forman una asociación público-privada que ofrece soluciones de nuevas arquitecturas y da unas normas para el desarrollo de la infraestructura del 5G. Esta asociación va a permitir a Europa seguir liderando en la creación de los nuevos mercados que se abren con la aparición de la quinta generación de la comunicación. También se pretende reforzar toda la industria para poder competir

con los mercados mundiales y así generar innovación en el sector tecnológico. Los principales retos de esta asociación con el 5G son los siguientes:

- Proporcionar un área inalámbrica 1000 veces mayor y con más capacidad de servicio.
- Ahorrar cerca del 90 % de energía, debido a que la única energía que se consumirá provendrá de la red de acceso radioeléctrico.
- Reducir el tiempo en la creación de un nuevo servicio a los 90 minutos.
- Crear una internet que sea segura, fiable y confiable en la prestación de los servicios hacia los usuarios de la Internet.
- Garantizar a todos los seres humanos y en todas las partes del mundo el acceso a una amplia gama de servicios y de aplicaciones a bajo coste.

Esta nueva generación de comunicaciones se ha desarrollo en varias fases, las cuales tenían objetivos marcados por 5GPPP [3]. La primera fase del desarrollo se basa en la investigación sobre los nuevos conceptos en los que se iba a desarrollar el 5G. Estas investigaciones se empezaron en el año 2010. La segunda fase del desarrollo se centro en la implementación de las plataformas investigadas en la primera fase. En la última fase del desarrollo se centra en hacer pruebas en las diferentes plataformas que han sido implementadas y desarrolladas en la segunda fase. Un ejemplo de las pruebas que se realizaron con el 5G en la sociedad fue en los juegos olímpicos de invierno de Corea en el 2018, también se iban a realizar pruebas en los juegos olímpicos de Tokio 2020 y en la EURO 2020, pero estas últimas pruebas no se pudieran dar debido al aplazamientos de dichos eventos debido a la pandemia provocada por la COVID-19.

En la figura 2.1 se puede apreciar un diagrama en el tiempo sobre las diferentes fases de desarrollo del 5G, sobre las distintas versiones que ha ido teniendo y sobre las diferentes pruebas que se realizaron.

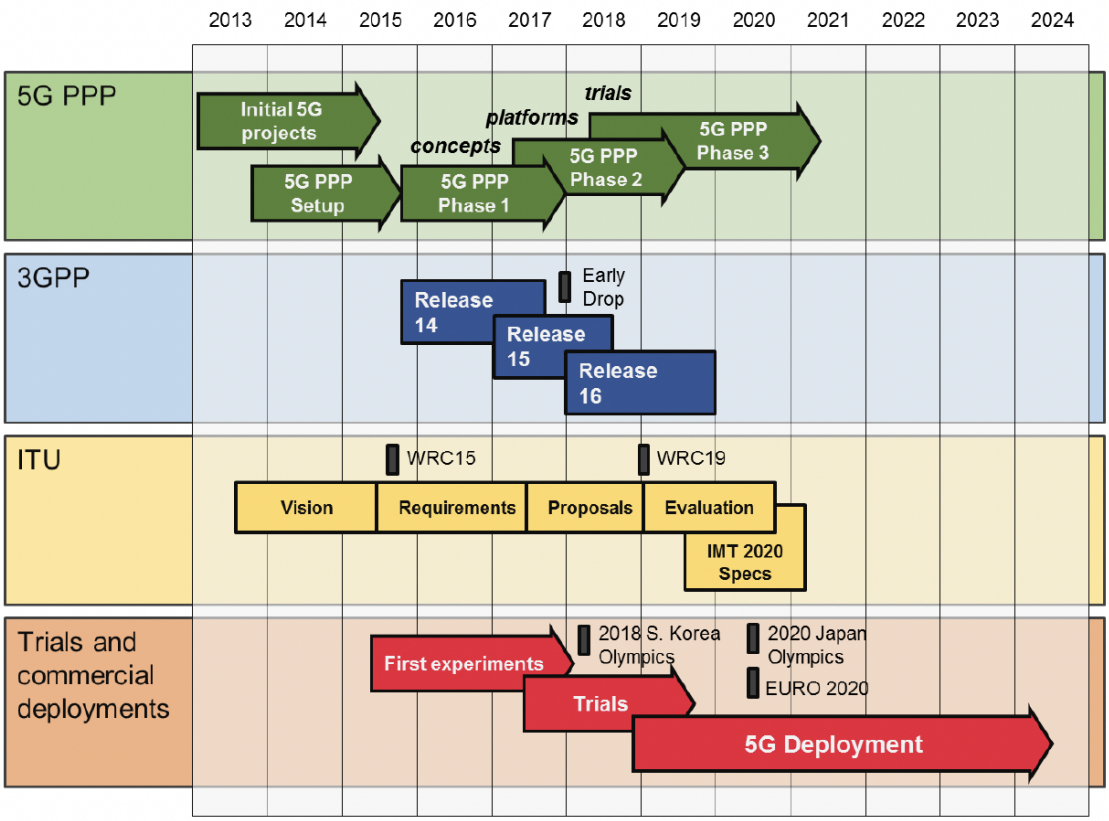

Figura 2.1: Desarrollo del 5G

Como se puede observar en la parte superior de la figura 2.1 nos muestra el camino seguido por la asociación 5G PPP en el desarrollo del 5G. En la segunda fila de la figura 2.1 se muestra las diferentes versiones que ha impuesto la 3GPP para la quinta generación de las comunicaciones. En la última fila de la imagen se puede ver cuando empezaron a hacer las primeras pruebas que se hicieron con el 5G y, también se puede ver en el año en el empieza el despliegue del 5G en la sociedad.

A continuación, se va a hablar de los diferentes proyectos que se llevaron a cabo la 5G PPP sobre el 5G:

- 5G-Crosshaul: es una red integrada de transporte de carga y descarga que permite una reconfiguración flexible y configurada por el software de todos los elementos de la red.
- 5GEx: permitir la orquestación de servicios en varios dominios.
- 5G-NORMA: es un desarrollo de una red 5G novedosa, adaptable y preparada para la futura arquitectura que tiene como objetivo el apoyo multi-arrendamiento y multiservicios.
- 5G-Xhaul: es una solución de red óptica e inalámbricas que es capaz de conectar pequeñas células a una red central.
- COHERENT: es un desarrollo sobre el control que se puede programar que sea unificado la coordinación y la gestión flexible de las redes de acceso heterogéneo del 5G.
- CHARISMA: se ha centrado en el enrutamiento jerárquico y virtualizado, que se une una descarga descentralizada y la cadena de servicios de seguridad extremo a extremo.

- FANTASTIC 5G: es una interfaz flexible y escalable con un diseño completo de PHY, MAC y RRM.
- Flex5GWare: son plataformas de hardware y software altamente reconfigurables que se dirigen a los elementos de la red y a los dispositivos.
- METIS-II: es un desarrollo general del 5G-RAN centrándose en la integración de los AIV y el apoyo de la división de la red.
- mmMAGIC: es un desarrollo sobre conceptos nuevos de la arquitectura RAN para ondas milimétricas de acceso a la tecnología de radio.
- Selfnet: es un marco de gestión de redes autonómicas para lograr la autoorganización de las capacidades en la gestión de la infraestructura de la red y mitigando los errores más frecuentes.
- Speed-5G: es la investigación de la gestión de recursos a través de "silos"tecnológicos y nuevas tecnologías de acceso medio para abordar la densificación de entornos en su mayoría no planificados.

2.1.3. Versiones del 5G

Como se ha podido observar en la Figura 2.1 durante el desarrollo del 5G ha tenido diferentes versiones, dónde estas versiones han sido impuestas por la asociación llamada 3GPP. La 3GPP [4] es una asociación colaborativa de organizaciones de telecomunicaciones, dónde su objetivo primordial es asentar las especificaciones de los sistemas de comunicaciones que se desarrollan. En esta asociación se dividen en tres grupos de desarrollo, los cuáles son:

- El grupo de las Redes de Acceso Radioeléctrico (RAN).
- El grupo de los aspectos de Servicios y Sistemas (SA).
- El grupo de la Red Central y Terminales (CT).

Todos estos grupos se reúnen con regularidad para presentar sus trabajos y debatir ellos y, así poder aprobar nuevas especificaciones sobre las redes de comunicaciones.

La 3GPP fue la encargada de desarrollo de las especificaciones sobre la nueva generación de las comunicaciones, la primera versión que lanzó al mercado la asociación fue la llamada **Relesase 15**, esta versión abarca las siguientes especificaciones y fue lanzada a mediados del año 2017:

- Los modelos que iban a tener los canales que se utilizarán en el 5G.
- La arquitectura modularizada del E2E, las varias opciones para la integración de eLTE/NR, las formas de control y del plano del usuario y, por último, las divisiones horizontales de la RAN.
- La arquitectura del QoS.
- La elección de la onda a utilizar, la codificación, la multi-antena, el soporte de formación de haces y la estructura del marco básico.
- La introducción al nuevo CRR y las optimizaciones de señalización.

Pasado un tiempo y después de varias reuniones de la asociación decidieron lanzar las versiones **Release 16** y **Release 17**, estas versiones se lanzan debido a los diferentes cambios que sufren el 5G en la fase de desarrollo. Estas versiones lanzadas abarcan las siguientes especificaciones:

- El auto-retorno, que es utilizar la misma interfaz radioeléctrica y el mismo espctro para el retorno.

- La extensión de la RN hacia el soporte de las slices en la red.
- La mejora de los medios de seguridad y la arquitectura conexa para el 5G.
- Gestión y orquestación del 5G.
- Posibles extensiones de forma de onda para servicios específicos de URLLC y mMTC.
- Acceso asistido para permitir el funcionamiento del NR en bandas sin licencia por encima de 6Ghz.
- El diseño del nuevo canal de acceso aleatorio (RACH).
- Agrupación de dispositivos para el acceso al sistema conjunto.
- Mejora del D2D.

Después de lanzar las anteriores versiones introdujeron la versión llamada **Longer-term and/or propietary**, que son especificaciones de larga duración y que se pueden aplicar de forma propietaria, es decir, una versión estable en el tiempo del 5G, dónde abarca estas especificaciones:

- La instanciación de la funcionalidad de la red para el apoyo multi-arrendamiento o multiservicio.
- La integración y la optimización conjunta del backhaul y del fronthaul.
- La automatización de la seguridad.
- La orquestación en los escenarios multi-dominios y de la multi-tecnología.
- La clasificación de servicios basadas en el aprendizaje automático.
- La dirección proactiva del tráfico que nos puede proporcionar la evaluación de los enlaces para reducir fallos en los enlaces.
- La gestión de las topologías radioeléctricas dinámicas.
- La gestión de múltiples slices.
- La salida múltiple masiva de entrada múltiple masiva que significa tener un gran número de antenas en el lado del transmisor como en el del receptor.
- Las consideraciones detalladas de las implementaciones que debe tener el hardware y el software.

A continuación, se mostrará la figura 2.2 sobre como las diferentes versiones que se han lanzado se han basado en las versiones anteriores y han añadido nuevas especificaciones para hacer el 5G más estable y que no tenga ningún fallo cuando se despliegue en la sociedad.

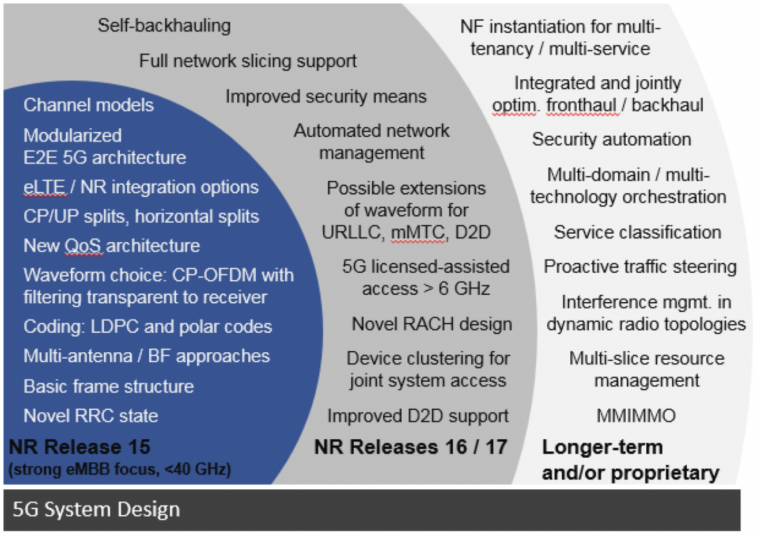

Figura 2.2: Versiones del 5G

2.2. Network Slicing

2.2.1. Introducción al Network Slicing

Para entender como funciona o qué es una slice dentro de la red, se tiene que enteder que el 5G prometió mejorar el ancho de banda en los servicios móviles, pero también se habilita el soporte a nuevos servicios como las llamadas las industrias verticales. Una de las claves para entender el network slice es ver los requisitos operacionales de los diferentes indicadores clave de rendimiento (KPI), los cuáles son:

- La velocidad de los datos que experimentan los usuarios.
- La latencia en los extremos.
- La fiabilidad.
- La eficiencia de la comunicación.
- La disponibilidad.
- la energía consumida.

Todos estos requisitos pueden ser satisfechos por entornos específicos, que esta es la funcionalidad de una network slice dentro de una red.

El **network slicing** es una de las claves para tener la flexibilidad en las redes 5G. Estas networks slices son necesarias para crear múltiples redes lógicas que están dentro de una misma infraestructura.

La funcionalidad de una network slice tiene muchas similitudes a soluciones que hay en la actualidad, estas soluciones son las siguientes:

- **VLANs**: son redes con diferentes hosts que están bajo el mismo dominio de broadcast.

- **VPNs**: son redes que se usan para conectar múltiples hosts mediante túneles que son privados y seguros.
- **CNs**.

Las soluciones anteriores no abordan todos los diferentes casos que pueden tener los **KPIs**. Una descripción de una **network slice** es que es una red lógica que proporciona las capacidades de una red específica y de una red característica. Otra definición es que es una red personalizada creada por un operador para dar una solución optimizada a un escenario específico del mercado, dónde nos exige unos requisitos que tengan un alcance de extremo a extremo.

La red se implementa con unas instancias llamadas **slices**. Estas **slices** usan las mismas funciones y pueden intercambiar información entre las diferentes slices que se han implementado en la red. Las slices de una red permite reducir el coste de desplegar una red y reducir el número de operaciones dentro de la red. Para su utilización se introducen soluciones de software, que estas soluciones están basadas en la virtualización y la modularización funcional. Las redes lógicas que se crean tienen una composición de una **red central** (CNF) y de las **funciones de red de radio** (RNF), dónde van a convivir todas dentro de la misma infraestructura. La red que va a soportar las diferentes redes lógicas que hay en su interior deben tener los siguientes elementos:

- La función de red de virtualización (NFV).
- El software de una red definida (SDN) dónde va a separar el plano de control (CP) y el plano de usuario (UP), la cuál, nos va a permitir la programabilidad de la red.

El concepto de una slice es el de una red lógica E2E que quiere proporcionar flexibilidad a la infraestructura de red.

La división en slices de una red tiene como objetivo principal reutilizar los recursos físicos y de software que tiene la infraestructura de red. Estas slices deben estar protegidas del rendimiento de las otras slices que se están ejecutando en la misma infraestructura, dónde el aislamiento se puede conseguir de dos formas distintas:

- Con la separación horizontal de recursos.
- Con mecanismos eficientes de planificación y coordinación entre todos los recursos de la red.

2.2.2. Soft-Hard Slicing

Introdución al Soft-Hard Slicing

Como se ha visto anteriormente el 5G se define por las distintas slices [5] que puede haber dentro de una infraestructura de red. Las slices pueden ser de dos tipos: soft y hard. La denominación a la slice es **Standarized Slice Type** (SST) dónde determinan como se asignan los recursos a nivel de RAN o a nivel de MC. Dentro de las SST hay diferentes tipos que se utilizan para diferentes aplicaciones, a continuación, se mostrará los diferentes tipos de SST y sus aplicaciones en el 5G:

- Las SST del tipo 1 se aplican para las aplicaciones basadas en el eMBB.
- Las SST del tipo 2 se aplican para las aplicaciones que están basadas en el URLLC.
- Las STT del tipo 3 se aplican para las aplicaciones que se basan en el mMTC.

Todas estas divisiones de red están disponibles para todos los componentes de las redes basadas en el 5G. Las divisiones de red pueden ser de dos tipos:

- **Soft-Network slicing**: se basa en los QoS, es decir, la red asigna dinámicamente recursos a la slice dependiendo del tráfico.
- **Hard-Network slicing**: se basa en el aprovechamiento del todo el software de la red y de las nubes 5G. Estas slices se logran mediante la virtualización y la replicación de los componentes 5G.

Soft network slicing

Las redes con divisiones soft [6] se basan asignando un índice de clase de los QoS a cada clase de tráfico. Cada componente de la red aplica los QoS de acuerdo con el mecanismo basado en el valor del QCI del flujo del tráfico. La finalidad de este tipo de slices es determinar la asignación de los recursos a nivel de RAN y a nivel de MC. Estas slices se crean de la siguiente manera: la estación base de la red crea uno o más túneles con el plano del usuario a nivel de MC. Estos túneles, que ha creado la estación base, garantizan el tráfico que pasa a través de ellos. Cada uno de estos túneles tienen QCI diferentes.

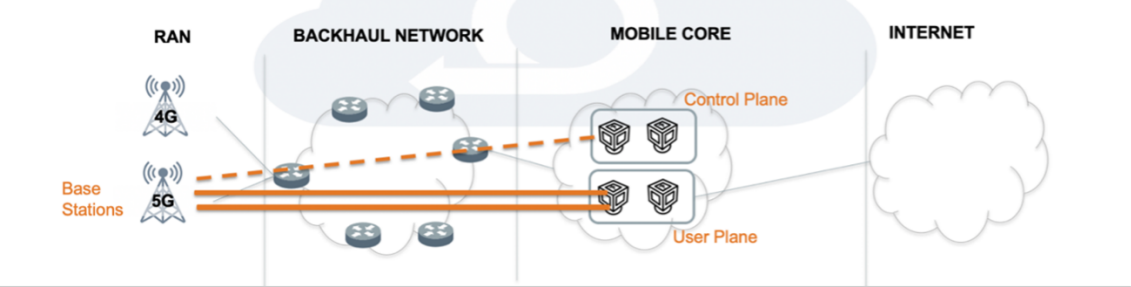

Figura 2.3: Soft network slicing

Estos tipos de slices se pueden dar por las primitivas de QoS en los nodos de la red IP, dónde estos nodos incluyen la clasificación de paquetes IP basado en el campo TOS/DSCP en la cabecera IP. Este tipo de slice no solo proporcional calidad de tráfico a nivel de tráfico también se pueden hacer a nivel de cliente.

Las redes que se basan en slcies del tipo soft-network slicing solo pueden garantizar la latencia hasta el **Mobile Core**, pero a partir del siguiente tramo que es el de **Internet** no se podría garantizar. Una solución para poder garantizar la latencia en **Internet** es acercar el DC al **Mobile Core** y al usuario final, pero esta solución sería necesario para las aplicaciones basadas en URLLC. Estos mecanismos no son suficientes para garantizar que las aplicaciones puedan satisfacer los requisitos necesarios para las aplicaciones 5G.

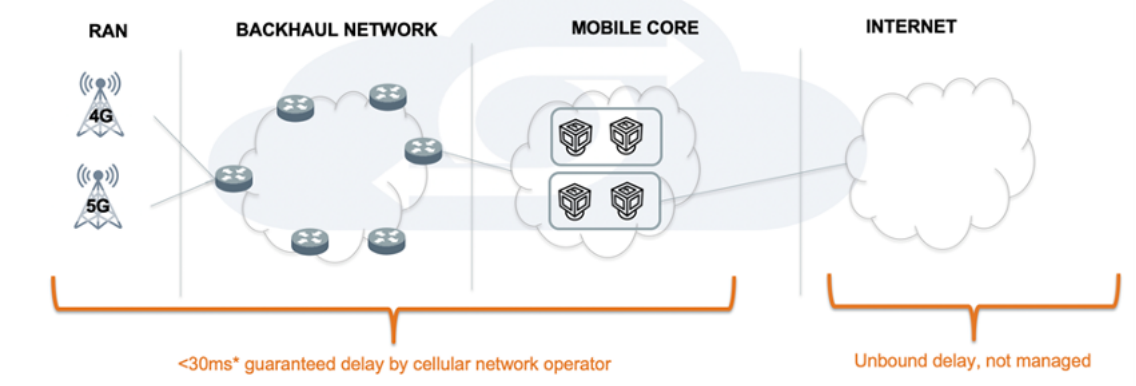

Figura 2.4: Delay Soft network slicing

Hard network slicing

Las redes del tipo hard-network slicing [7] se basan en la capacidad del 5G para virtualizar y replicar los componentes de la red para poder tener instancias separadas con diferentes necesidades, todo esto es gracias a la naturaleza del software y a las nubes de arquitectura del 5G. Los componentes que integran la red se dividen en componentes virtuales que pueden ser replicados y ejecutados en diferentes lugares de la red. La malla que se forma en el **Mobile Core** del la red 5G puede ser replicada y con esa replicación se pueden crear múltiples MC de diferentes tipos. Con la creación de múltiples MC se pueden tener en diferentes lugares de la red, como, por ejemplo:

- Para las aplicaciones basadas en URLLC se pueden ejecutar en el centro de datos del extremo.
- Para los otros tipos de aplicaciones se pueden ejecutar en zonas más centrales de la red.

Como se ha visto anteriormente las redes backhaul se basan en la división entre el plano de datos y el plano de control, dónde en el plano de datos funciona con un hardware especializado, mientras en el plano de control funciona con la replicación y virtualización de la nube, por esta razón se pueden crear múltiples routers virtuales dentro del mismo dispositivo físico.

La combinación de una red basada en el hard-network slicing y el Mobile Edge Computing (MEC) permite a los operadores poder crear y gestionar múltiples redes virtuales y tener la flexibilidad de poder asignar estas redes virtuales a los diferentes tipos de aplicaciones que se pueden desarrollar. Estas redes virtuales que se han creado en la misma infraestructura soportan los requisitos heterogéneos de las diferentes aplicaciones. Uno de los problemas de este tipo de red 5G es que la comunicación entre la red virtualizada y la red backhaul solo podría ser compatible con mecanismos tradicionales de comunicación a bajo nivel. Una solución a este problema y tener una hard-network slicing de extremo a extremo es que la red backhaul pueda soportar la replicación y la virtualización.

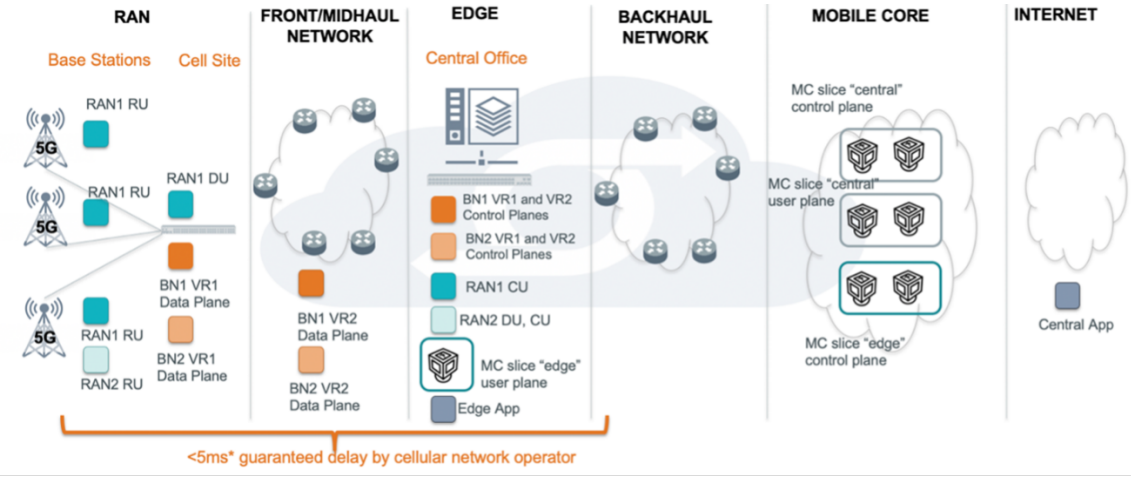

Figura 2.5: Hard network slicing

3

Diseño

Este trabajo fin de grado consiste en realizar un Network Slice Controller (NSC). Este NSC está formado por tres elementos principales: (1) Network Slice Controller North-Bound Interface (NSC-NBi), (2) la base de datos de estado y (3) Network Slice Orchestrator. El NSC recibe llamadas desde un cliente que este dentro de una red 5G para poder modificar sus necesidades dentro de la red dependiendo de las aplicaciones que este ejecutando en ese momento. Se ha llamado a este cliente: Aplicación.

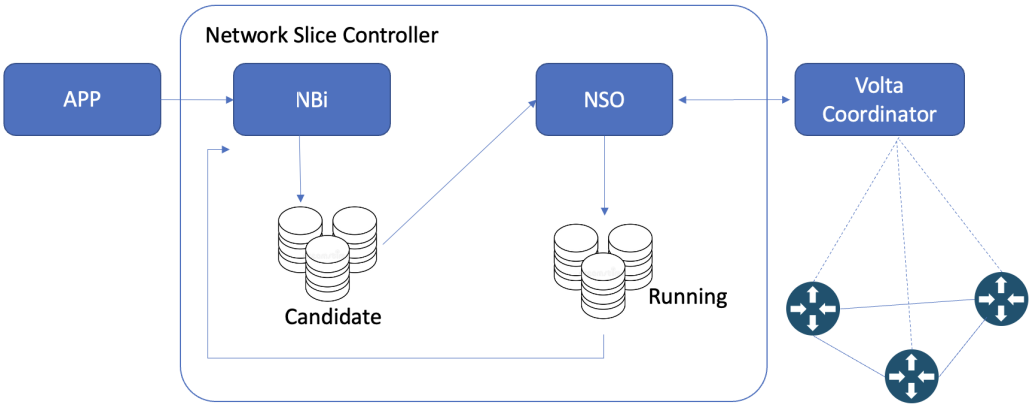

Figura 3.1: Diseño simplificado del proyecto

Como se puede observar en la figura 3.1, el diseño completo de la aplicación, dónde se explicará a continuación que función tiene cada elemento de este gráfico y el porqué se ha decidido desarrollarlo así.

3.1. Network Slice Controller North-Bound Interface

En primer lugar, se va a explicar en que consiste el **Network Slice Controller North-Bound Interface (NSC-NBi)**. Este elemento corresponde a una API que es la encargada de llamar exponer las capacidades de configurarción de la red para que pueda cumplir con las necesidades que tenga la aplicación de 5G. Esta API va a tener los comandos CRUD, a continuación, se explicará que hacen

estos comandos en nuestra aplicación:

- **Create**. Si la aplicación realiza un POST al servidor, el NSC-NBi va a enviar una nueva configuración de red al servidor.
- **Read**. Si la aplicación realiza un GET al servidor, el NSC-NBi va a obtener la configuración que se esta ejecutando en ese momento.
- **Update**. Si la aplicación realiza un PUT al servidor, el NSC-NBi va a enviar a la aplicación la nueva configuración actualizada, cambiando la última configuración que se haya enviado.
- **Delete**. Si la aplicación realiza un DELETE, el NSC-NBi se borrará la última configuración enviada al NSC.

El modelo YANG que se ha utilizado para crear el NSC-NBi es el siguiente: **draft-liu-teas-transport-network-slice-yang-02** [8]. Este modelo YANG define un modelo de datos para poder representar, gestionar y controlar slices dentro de una red 5G. Este modelo es una interfaz cliente-proveedor para poder realizar configuraciones y recuperaciones de las slices de red. Con este modelo se puede actualizar los requisitos que tenga el cliente para asi poder hacer una configuración a medida de una slice de la red. Este modelo que se ha diseñado para dar múltiples opciones de slicing, las cuales son:

- **Slicing de red por virtualización**. Este tipo de aplicación realiza una virtualización de los routers para que en un mismo router pueda convivir dos infraestructuras de redes distintas. Como se muestra en la figura 3.2. Tal y como se puede ver en misma, el modelo permite realizar la virtualización de seis routers para dos slices de clientes (azul y roja). Esto es posible creando routers virtuales (VR) sobre los routers físicos.

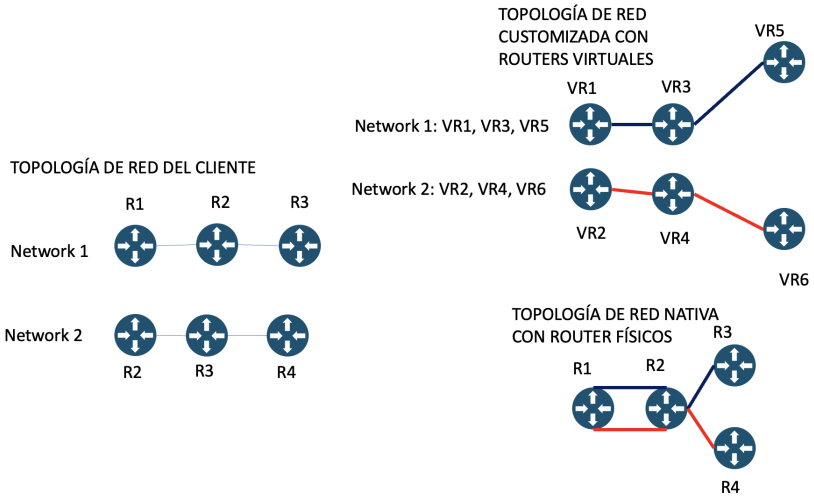

Figura 3.2: Slicing de red por virtualización

- **Slicing de red por superposición de ingeniería de tráfico**. Este tipo de aplicación realiza un túnel independiente por cada red y cada túnel viaja la información de la red sin perjudicar a los otros túneles que haya en la infraestructura de la red. La figura 3.3 muestra este modelo de slicing donde existen dos slices de clientes (azul y roja) y seis routers, pero en este caso se crean túneles de ingeniería de tráfico para ofrecer a cada uno las características que requiere el servicio. El tráfico del cliente de la slice roja irá por unos túneles diferentes a los del cliente de la slide azul.

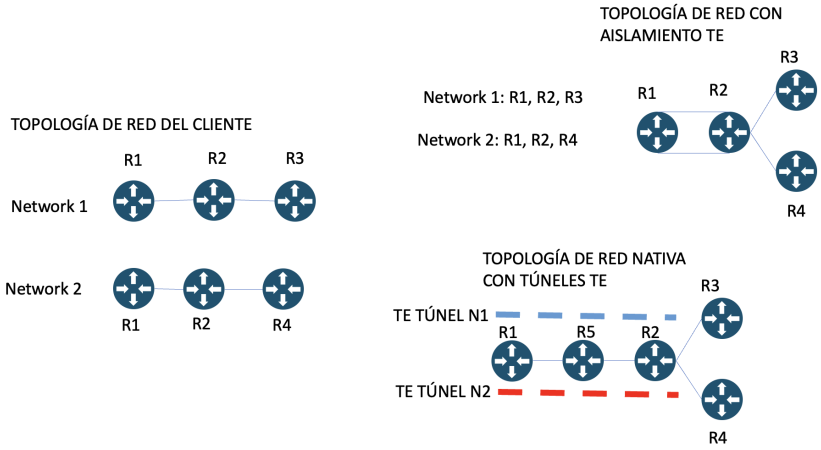

Figura 3.3: Slicing de red por superposición de TE

Se ha elegido este diseño debido a que el proyecto tiene que realizar virtualizaciones de los routers de la red para crear diferentes slices dentro de una misma infraestructura de red.

El lenguaje elegido para realizar el NSC-NBi es Python, debido a que es un lenguaje sencillo de entender y de fácil programación para el modelo elegido del proyecto.

3.2. Base de datos de estado

Las dos bases de datos diseñadas en este proyecto han seguido el diseño propuesto en la **RFC 6241** [9]. La RFC 6241 define el mecanismo para que un dispositivo de red pueda administrar, recuperar o manipular información de las configuraciones de red. Este diseño permite que se pueda reducir costes en fase de implementación y dar un acceso oportuno a los datos que hay almacenados en las distintas bases de datos. En esta RFC 6241 se definen varios estados de las configuraciones, de los cuales dos han sido introducidos en el diseño del proyecto. Estos dos estados son:

- **Running**: Este término se ha define en la RFC 6241 como un almacén de datos que contiene la configuración, que está activa en el red. El almacén de datos de configuración en ejecución siempre debe existir.
- **Candidate**: Este término se ha definido dentro de la RFC como un almacén de datos de configuración que puede ser manipulado sin afectar a la configuración actual, que se este ejecutando, y que puede ser almacenada en el almacén de datos de de datos de configuración Running. Se tiene que tener en cuenta que una configuración candidata puede estar compartida por varias sesiones. Por lo tanto, si un cliente quiere modificar una configuración

debe bloquear el recurso Candidate antes de que modifique su configuración, porque puede haber un problema de incongruencia de datos.

Dentro del NSC, se ha creado una base de datos no relacional llamada **Candidate**, esta base de datos en una base de datos NoSQL Redis. La función que tiene esta base de datos en nuestro diseño es la de almacenar todas las posibles configuraciones de la red que se han enviado al servidor desde nuestra API.

También, se creó la base de datos NoSQL Redis llamada **Running** que se ha creado para guardar todas las configuraciones de red que han sido ejecutadas por el **Volta Coordinator** para tener un resumen de las diferentes configuraciones que han sido ejecutadas.

Como podemos ver en la figura 3.1 tiene dos productores-consumidores, esto es debido a que puede haber fallos de concurrencia con las lecturas en las distintas bases de datos que tengo en el proyecto. El primer productor-consumidor es el que se realiza en el **NSC-NBi** dónde produce todas las configuraciones necesarias para la base de datos **Candidate**, pero a su vez tiene que consumir todas las configuraciones necesarias en la base de datos **Running**, debido a que este es el ciclo que debe seguir la información, pero a su vez hay otro Productor-Consumidor que es el que hay entre **Candidate** y el **Network Slice Orchestartor**, que se explica en el siguiente punto.

3.3. Network Slice Orchestrator

La funcionalidad del **Network Slice Orchestrator** se ha basado el siguiente draft: **draft-barguil-teas-network-slices-instantation-00** [10] y así poder realizar un NSO genérico para todas las posibles configuraciones de red que pueden llegar a este elemento.

En este draft muestra la función del Network Slice controller para realizar la traducción de los modelos genéricos de slices con los diferentes modelos de la red. El modelo YANG utilizado en este trabajo fin de grado, permite tener una idea de como esta orientada la red. Por lo tanto, el Network Slice Controller recibe la solicitud de creación de una slice de red y en este draft muestra expandir los parámetros específicos para que puedan ser utilizados por el controlador de la red. De este modelo se ha utilizado en el que el Network Slice Controller sea un elemento independiente del controlador de la red, como se puede observar en la figura 3.4.

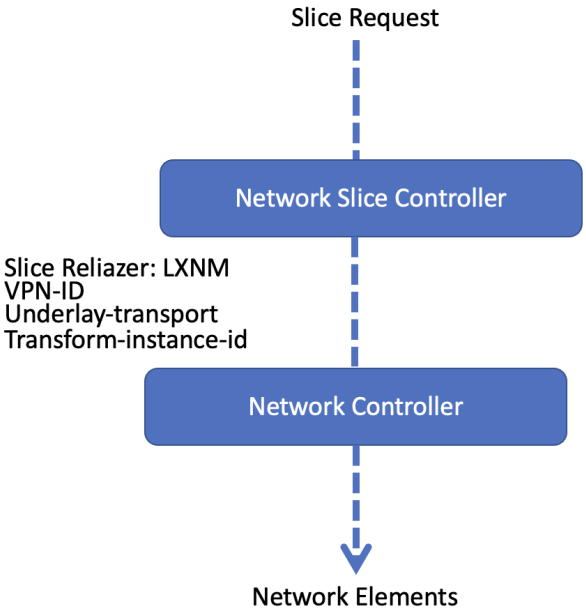

Figura 3.4: Diseño de un Network Slice controller independiente del Netowrk Controller

Este draft muestra también a realizar un tratamiento de la petición de creación o modificación de una slice de red, dónde se han de seguir los siguientes pasos:

- Primero se ha de mapear, es decir, procesar la solicitud del cliente.
- Segundo se ha de crear todas las solicitud necesarias a la red para realizar la slice en la red.

En nuestro proyecto se ha creado el **Network Slice Orchestrator** siguiendo el diseño del draft comentado anteriormente. El NSO es un módulo dentro del Network Slice Controller que traduce las llamada de Slices en parámetros de red. El NSO habla con un Network Controller que es el **Volta Coordinator** en este trabajo. Internamente, el NSO es un proceso que se está ejecutando todo el rato dentro de la red y la función principal es la de aceptar o rechazar las distintas configuraciones que tiene la base de datos **Candidate**. El **Network Slice Orchestrator** realiza comprobaciones sobre las configuraciones que están en **Candidate**, al comprobar que la sintaxis del JSON de configuración es correcta llama en primer lugar a **Volta Coordinator** con diferentes llamadas para poder configurar la red a las necesidades que ha propuesto un equipo desde la API. En este caso, no existe una API standard como se sugiere en el draft, sino que la adaptación se hace a la API usada por el **Volta Coordinator**. En segundo lugar, escribe la configuración que ha pasado al **Volta Coordinator** en una base de datos no relacional llamada **Running**. El productor-consumidor entre el **NSO** y la base de datos de estados **Running** es necesario para poder pasar las configuraciones candidatas al Orchestrator y que las pueda enviar al **Volta Coordinator** y así poder configurar la red a nuestras necesidades y después guardar la configuración en **Running**. La ventaja de usar estos dos productores-consumidores es que las configuraciones siguen el ciclo que se ha marcado y así no tenemos problemas de concurrencia

con los datos leídos y enviados al **Volta Coordinator**.

4

Desarrollo

El desarrollo que se ha realizado para el proyecto ha tenido el siguiente ciclo de vida, dónde primero se ha hecho una primera fase de investigación sobre el 5G y el Network Slicing y sobre los diferentes documentos del IETF que se han podido utilizar para la creación del NSC-NBi. Al realizar este estudio previo se ha escogido el draft que mejor convenia al proyecto para poder crear el NSC-NBi para nuestras necesidades.

La segunda fase del ciclo de vida fue desarrollar el proyecto entero para su funcionamiento y posteriormente para la realización de pruebas de este. Dentro de esta fase del desarrollo del proyecto se ha divido en cuatro partes. Estas cuatro partes están bien diferenciadas debido a que cuando tenía la primera fase del desarrollo se ha probado para corregir errores y así poder seguir avanzando y, así lo se ha hecho con las diferentes fases de este proyecto. Para tener un gran control sobre las diferentes fases, se ha creado un repositorio de git con diferentes ramas, el cual ha servido de punto de partida para poder ir desarrollando la siguiente fase del proyecto. A continuación, se mostrará la figura 4.1 de como ha ido evolucionando la funcionalidad del proyecto.

Figura 4.1: Funcionalidad del proyecto

4.1. Fase de desarrollo Network Slice Controller North-Bound Interface

En esta fase se ha desarrollado el Network Slice Controller North-Bound Interface (NSC-NBi) que permite la comunicación con el APP. A continuación, se va a describir los diferentes pasos realizados para la creación del NSC-NBi:

En primer lugar, es descargar todos los modelos YANG necesarios para poder crear el modelo YAML para poder crear nuestro NSC-Nbi. Los modelos YANG necesarios para poder realizar el NSC-Nbi son los siguientes: *ietf-inet-types, ietf-network-slice, ietf-network-topology, ietf-network, ietft-routing-types, ietf-te-types* e *ietf-yang-types*. Cuando nos descarguemos todos esos modelos YANG tendremos que ejecutar el siguiente comando: **pyang -f tree ietf-network-slice.yang**, todos los modelos YANG deberán estar en el mismo directorio para que se pueda ejecutar bien el anterior comando. Si se han descargado todos los modelos YANG necesarios para poder crear el NSC-Nbi, aparecerá la siguiente información en la consola.

```
module: ietf-network-slice

  augment /nw:networks/nw:network/nw:network-types:
    +--rw network-slice!
  augment /nw:networks/nw:network:
    +--rw network-slice
       +--rw optimization-criterion?   identityref
       +--rw delay-tolerance?          boolean
       +--rw periodicity*              uint64
       +--rw isolation-level?          identityref
  augment /nw:networks/nw:network/nw:node:
    +--rw network-slice
       +--rw isolation-level?   identityref
       +--rw compute-node-id?   string
       +--rw storage-id?        string
  augment /nw:networks/nw:network/nt:link:
    +--rw network-slice
       +--rw delay-tolerance?   boolean
       +--rw periodicity*       uint64
       +--rw isolation-level?   identityref
```

Figura 4.2: Comando Pyang sobre el draft

Si no se han descargado bien algún modelo, este comando dará información de que modelo YANG esta fallando para que se pueda corregir el modelo y así poder crear el NSC-Nbi. Después de comprobar que se no habido un comando en el comando **pyang**, se va a realizar la creación del NSC-Nbi YAML, para poder crear el NSC-NBi se deberá ejecutar el siguiente comando:

java -jar swagger-generator-cli-1.1.15-SNAPSHOT-executable.jar -yang-dir . -fullCrud false output <nombre-fichero>.yaml -content-type application/yang-data+json <modelos-yang>

Dónde este comando sacará un archivo YAML con las diferentes llamadas que tendrá el NSC-NBi. Después de obtener el archivo YAML se deberá ejecutar el siguiente comando:

java -jar swagger-codegen-cli-2.2.1.jar -generate <nombre-fichero>.yaml -l python-flask

Dónde le diremos que lenguaje tiene que utilizar para crear el NSC-Nbi, en este caso se ha elegido

que se cree un NSC-NBi en python con el micro-framework **Flask**. Al ejecutar este comando creará los archivos necesarios para poder ejecutar un proceso NSC-Nbi y poder realizar llamadas sobre él.

Después de realizar los siguientes pasos, ya se podría ejecutar nuestro proceso NSC-Nbi con el siguiente comando:

python3 app.py

Y ya se estará ejecutando el proceso NSC-Nbi para atender todas las llamadas que se hagan desde la APP.

En esta fase de desarrollo se ha escrito las siguientes líneas de código:

Código 4.1: En esta figura se presenta el código correspondiente a las funciones básicas que tiene el NSC-Nbi YAML.

```
from flask import request

networks = {}
def data_networks_delete() -> str:
    if networks != None:
        networks.clear()
        return "Object deleted"
    else:
        return "Internal error", 204

def data_networks_get() -> str:
    if networks == None:
        return "Internal error", 400
    elif networks == {}:
        return networks, 200
    else:
        return networks['networks'], 200

def data_networks_post(ietf_network_Networks_bodyParam = None) -> str:
    json = request.json
    if json == {}:
        return "Internal error", 400
    else:
        networks['networks'] = json
        return "Object created", 201
```

Como se puede observar, se ha creado un diccionario para guardar las diferentes configuraciones que pueda enviar la APP. También se observa los diferentes códigos de respuesta que puede tener el NSC-Nbi.

Ahora se va a explicar el desarrollo seguido para la creación del NSC-NBi. En este desarrollo se han hecho varias funciones que crear el JSON que se envía a nuestro NSC-Nbi para que realice el almacenamiento de la configuración.

Código 4.2: En esta figura se presenta el código correspondiente a la función de creación de un nodo.

```
1  def crear_nodo_network(network_id, network_point, network_type):
2      dicc = {}
3      termination_point = {}
4      network_types = {}
5
6      if network_id != "" and network_point != "" and network_types != "":
7          dicc["node-id"] = network_id
8          network_types["isolation-level"] = network_type
9          dicc["ietf-network-slice:network-slice"] = network_types
10         dicc["ietf-network-topology:termination-point"] = []
11         termination_point["tp-id"] = network_point
12         dicc["ietf-network-topology:termination-point"].append(termination_point)
13     else:
14         print("Error en los parámetros de entrada.")
15
16     return dicc
```

En el código 4.2, se muestra la creación de un nodo en la configuración de la red, dónde se le pasa por parámetros el id, el tipo de red y el nodo terminal. Con estos tres parámetros se crea un diccionario con la configuración del nodo de la red.

Código 4.3: En esta figura se presenta el código correspondiente a la función de creación de un enlace

```
1  def crear_link_network(link_id, source_node, source_tp, dest_node, dest_tp, network_type):
2      dicc = {}
3      source = {}
4      dest = {}
5      network_types = {}
6
7      if link_id != "" and source_node != "" and source_tp != "" and dest_node != "" and dest_tp != ""
           and network_type != "":
8          dicc["link-id"] = link_id
```

Código 4.4: En esta figura se presenta el código correspondiente a la función de creación de un enlace

```
9            source["source-node"] = source_node
10           source["source-tp"] = source_tp
11           dest["dest-node"] = dest_node
12           dest["dest-tp"] = dest_tp
13           dicc["source"] = source
14           dicc["destination"] = dest
15           network_types["isolation-level"] = network_type
16           dicc["ietf-network-slice:network-slice"] = network_types
17       else:
18           print("Error en los parámetros de entrada.")
19
20       return dicc
```

En el código 4.3 y en el código 4.4, se muestra la creación de un enlace en la configuración de la red, donde se le pasa por parámetros el id, el nodo fuente con su nodo terminal y el nodo destino con su nodo terminal.

Código 4.5: En esta figura se presenta el código correspondiente a la función de creación de una red

```
1  def crear_network (network_id, network_type):
2      dicc = {}
3      network_types = {}
4
5      if network_id != "" and network_type != "":
6          dicc["network-id"] = network_id
7          dicc["network-types"] = {"ietf-network-slice:network-slice": {}}
8          dicc["node"] = []
9          dicc["ietf-network-topology:link"] = []
10         network_types["isolation-level"] = network_type
11         dicc["ietf-network-slice:network-slice"] = network_types
12     else:
13         print("Error en los parámetros de entrada.")
14
15     return dicc
```

En el código 4.5 se crea el diccionario necesario para poder añadir más adelante los nodos y los enlaces de la red. Esta función se le pasa por parámetros el id de la red y el tipo de red que se quiere crear.

A continuación, se mostrará las dos funciones para añadir nodos y enlaces a la red.

Código 4.6: En esta figura se presenta el código correspondiente a la función de añadir un enlace a la configuración de la red

```
1  def anadir_link_network␣(network,␣link):␣
2
3      if network␣!=␣{}␣and link␣!=␣{}:␣
4          network["ietf-network-topology:link"].append(link)␣
5      else:␣
6          print("Error␣en␣los␣parámetros␣de␣entrada.")␣
7
8      return network␣
```

Código 4.7: En esta figura se presenta el código correspondiente a la función de añadir un enlace a la configuración de la red que se esta creando.

```
1  def anadir_endpoint_network(network,␣endpoint):␣
2
3      if network␣!=␣{}␣and endpoint␣!=␣{}:␣
4          network["node"].append(endpoint)␣
5      else:␣
6          print("Error␣en␣los␣parámetros␣de␣entrada.")␣
7
8      return network␣
```

Después de esta creación de la red, se han creado las diferentes funciones para poder llamar al NSC-Nbi.

Código 4.8: En esta figura se presenta el código correspondiente a las diferentes llamadas posibles que podemos hacer a nuestro NSC-Nbi.

```
1  def post_network_slice(bodyParam):␣
2      headers␣=␣{'content-type':'application/yang-data+json'}␣
3      body␣=␣json.dumps(bodyParam,␣indent=4)␣
4      ret␣=␣requests.post('http://localhost:8080/data/networks/',json=body)␣
5      return ret.json()␣
6
7  def get_network_slice():␣
8      ret␣=␣requests.get('http://localhost:8080/data/networks/')␣
9      return ret.json()␣
10
11 def delete_network_slice():␣
12     ret␣=␣requests.delete('http://localhost:8080/data/networks/')␣
13     return ret.json()␣
```

4.2. Fase de desarrollo Base de datos de estado

Después del desarrollo de la primera fase desarrollando el NSC-NBi, se comenzó la segunda fase del desarrollo añadiendo la base de datos no relacional creada con Redis llamada **Candidate**. Para la introducción de esta base de datos tenemos que modificar el NSC-NBi para que pueda introducir, eliminar o obtener las diferentes configuraciones que se han enviado desde la APP.

Código 4.9: En esta figura se presenta el código correspondiente a los controladores de las llamadas al NSC-NBi con las llamadas a la base de datos.

```
networks = {}
def data_networks_delete() -> str:
    if Candidate.getSizeOfCandidate() == 0:
        return "Object deleted"
    else:
        return "Internal error", 204

def data_networks_get() -> str:
    if Running.getSizeOfRunning() < 0:
        return "Internal error", 400
    elif Running.getSizeOfRunning() == 0:
        network = Running.getRunning()
        return network, 200
    else:
        network = Running.getRunning()
        return network, 200

def data_networks_post(ietf_network_Networks_bodyParam = None) -> str:
    json = request.json
    if json == {}:
        return "Internal error", 400
    else:
        Candidate.saveCandidate(json)
        return "Object created.", 201
```

Como se puede observar que hay diferentes cambios con respecto a la fase anterior, dónde se ha sustituido la creación del diccionario por diferentes llamadas a una API que tiene la base de datos **Candidate**.

Código 4.10: En esta figura se presenta el código correspondiente a las funciones de la API que tiene la base de datos Candidate.

```
1  redisClient = redis.Redis(host='localhost', port=6379, db=0)
2
3  def saveCandidate(networkSlice):
4      ret = redisClient.lpush('Network-Slice', networkSlice)
5      print(ret)
6
7  def getCandidate():
8      nts = redisClient.rpop('Network-Slice')
9      if nts == None:
10         return {}
11     networkSlice = json.loads(nts.decode("utf-8"))
12     return networkSlice
13
14 def getSizeOfCandidate():
15     size = redisClient.llen('Network-Slice')
16     return size
```

Como se puede observar en la primera línea se crea la conexión a la base de datos Redis. Después, se tiene la función que guarda la configuración en la base datos llamada **Network-Slice**. A continuación, se muestra la función con la cuál se puede obtener la primera configuración que se ha introducido en la base de datos. Por último, se tiene la función para obtener el número de configuraciones que hay en la base de datos, esta función es necesaria para poder hacer un control de errores a la hora de obtener las configuraciones.

4.3. Fase de desarrollo Network Slice Orchestrator

En esta fase del desarrollo se ha creado un proceso que se estará ejecutando llamado el **Network Slice Orchestrator**. Este proceso se encarga de validar la configuración de red solicitada en la base de datos **Candidate**. Se comprueba que la configuración no tiene ningún error de sintaxis y cuando este proceso devuelve que la configuración cogida de **Candidate** esta correcta, llamaría a Volta Coordinator, que en esta fase de desarrollo no estaría operativo, y esa misma configuración la introduciría en la base de datos **Running**. A continuación, se va a mostrar el código correspondiente a todas las comprobaciones que hace el **Network Slice Orchestrator** sobre la configuración red de la primera base de datos.

Código 4.11: En esta figura se presenta el código correspondiente a las comprobaciones que hace el proceso Network Slice Orchestrator

```
def checkNetworkSlice_():
    running = True

    if getCandidateOrchestrator == 0:
        print("Error in checkNetworkSlice.")
    networkSlice = Candidate.getCandidate()

    if len(networkSlice["ietf-network:networks"]) == 0:
        print("Error in checkNetworkSlice. Length is zero")
        return -1

    if "network" in networkSlice["ietf-network:networks"]:
        for network in networkSlice["ietf-network:networks"]["network"]:
            print("Network-id: "+ str(network["network-id"]))
            print("Nodes of Network:")
            if "node" in network:
                for node in network["node"]:
                    print("\t Node id: "+ str(node["node-id"]))
                    if "ietf-network-topology:termination-point" in node:
                        for tp in node["ietf-network-topology:termination-point"]:
                            print("\t Termination Point id: "+ str(tp["tp-id"]))
            else:
                running = False

            if "ietf-network-topology:link" in network:
                for link in network["ietf-network-topology:link"]:
                    print("\t Link id: "+link["link-id"])
                    print("\t Link Source Node:"+link["source"]["source-node"])
                    print("\t Link Source TP:"+link["source"]["source-tp"])
                    print("\t Link Destination Node:"+link["destination"]["dest-node"])
                    print("\t Link Destination TP:"+link["destination"]["dest-tp"])
            else:
                running = False
    else:
        running = False
        print("Error in Network Slice.")

    if running == True:
        print("******* AQUI VENDRÍAN LAS LLAMADAS A VOLTA. ********")
        sendToRunnig(networkSlice)
```

También se puede observar que este proceso saca por pantalla la configuración que se va a enviar al Volta Coordinator, después de realizar todas las comprobaciones necesarias.

La segunda parte de esta fase de desarrollo es crear la base de datos **Running**. Esta base de datos va ser una base de datos NoSQL, es decir no relacional, también se ha creado con Redis igual que la base de datos **Candidate**. Como se puede observar en el código posterior, tiene una API muy parecida a la API de **Candidate**. La funcionalidad es la misma, pero esta configuración se carga en **Running**, cuando se haya podido realizar la configuración real en los equipos de red.

Código 4.12: En esta figura se presenta el código correspondiente a la API necesaria para hacer operaciones en la base de datos Running.

```
1  def saveRunning(networkCandidate):␣
2      if networkCandidate␣==␣None:␣
3          print("Error␣in␣saveRunning.")␣
4          return -1␣
5      ret␣=␣redisClient.lpush('Running',␣networkCandidate)␣
6
7  def getRunning():␣
8      nts␣=␣redisClient.rpop('Running')␣
9      if nts␣==␣None:␣
10         return {}␣
11     networkRunning␣=␣json.loads(nts.decode("utf-8"))␣
12     return networkRunning␣
13
14 def getSizeOfRunning():␣
15     size␣=␣redisClient.llen('Running')␣
16     return size␣
```

4.4. Fase de desarrollo Volta Coordinator

En esta última fase de desarrollo se han incluido las llamadas al Volta Coordinator y también se ha añadido funcionalidad para crear tantos directorios como nodos que se haya creado en la configuración pasada al Network Slice Orchestrator. En estos directorios se guardan los ficheros necesarios para realizar las llamadas al Volta Coordinator. El rol del Volta Coordinator es configurar los routers de red con la configuración necesaria.

A continuación, se explicará los pasos necesarios que se han realizado para replicar la configuración de la red al Volta Coordinator [11].

- En primer lugar, se realiza la configuración del dispositivo. Las configuraciones que están permitidas son las siguientes:
    - Ajuste del hardware y los niveles de log del CP.

- Ajuste de la velocidad.
- Ajuste de los puertos de ESMC.
- Ajuste de las fuentes de Synce.

Para esta configuración utilizamos el siguiente comando: **Python device-setup.py –json-file <PATH-TO-JSON>** Todos los ajustes que se han comentado anteriormente están introducidos en el JSON que se utiliza en el comando anterior.

- En segundo lugar, se crean todas las interfaces necesarias para el dispositivo que se esta realizando la configuración con el siguiente comando:

  Python device-vlan-interfaces.py –json-file vlan-interfaces.json

- En tercer lugar, se debe realizar la configuración de los vrouters. Los usuarios pueden especificar con el tamaño del vrouter. Si el vrouter se configura con un contenedor Netconf, que en nuestro caso es así, se tiene que configurar las LIF, los brigdes, los VRF y las direcciones IP.

  Python device-vrouter.py –json-file <PATH-TO-JSON>–action CREATE

- Por último, se realiza la habilitación de los protocolos y servicios necesarios de los virtual routers. Los diferentes protocolos y servicios que se pueden habilitar son los siguientes:

  - BGP
  - ISIS
  - LPD

  Dónde los comandos necesarios para habilitar los servicios y protocolos son los siguientes:
  Python device-vrouter-bgp.py –json-file vrouter-bgp.json –CREATE Python device-vrouter-isis.py –json-file vrouter-isis.json –CREATE Python device-router-lpd.py –json-file vrouter-lpd.json –CREATE

Todos estos pasos se deben realizar tantas veces como dispositivos haya en la configuración de red que ha obtenido el Network Slice Orchestrator.

5

Integración, pruebas y resultados

En la siguiente figura 5.1 se va apreciar la implementación en el laboratorio del proyecto con sus respectivas direcciones IPs del NSC y del Volta Coordinator.

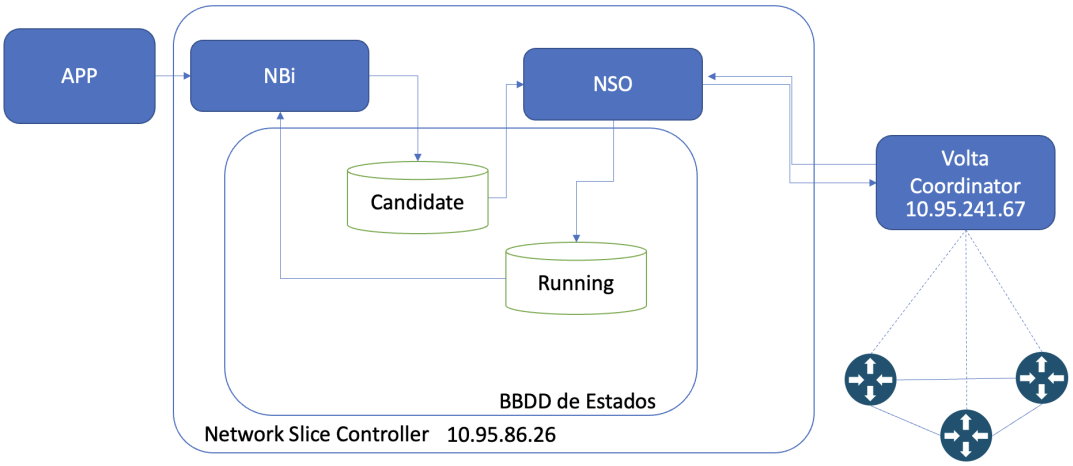

Figura 5.1: Diseño del proyecto en el laboratorio

Como se puede observar en la figura 5.1 se observa que la APP manda peticiones al Network Slice Controller. Dentro del **Network Slice Controller** está el NBi que es el encargado de recibir y almacenar las configuraciones de red en Candidate. Después de que el NBi almacene las configuraciones de red, el NSO es el encargado de obtener, realizar las comprobaciones necesarias de las configuraciones de red obtenidas de Candidate y alamcenar las configuraciones de red que se están ejecutando en Running. Por último, el NSO también es el encargado de realizar las llamadas a los scripts del Volta Coordinator para poder crear las configuraciones de red, que han sido solicitadas desde la APP.

Para todas las fases de desarrollo del proyecto se han realizado las mismas pruebas. La prueba más común es analizar las trazas de la APP con el servidor creado y ver si la configuración que se estaba realizando en la APP era correcto. A continuación, se mostrará las distintas llamadas que se realizaban al servidor con sus códigos de respuesta hacía nosotros, a parte, de visualizar las trazas que hemos grabado con **WhireShark** también veremos la salida del proceso a las distintas llamadas

que se hacen al servidor.

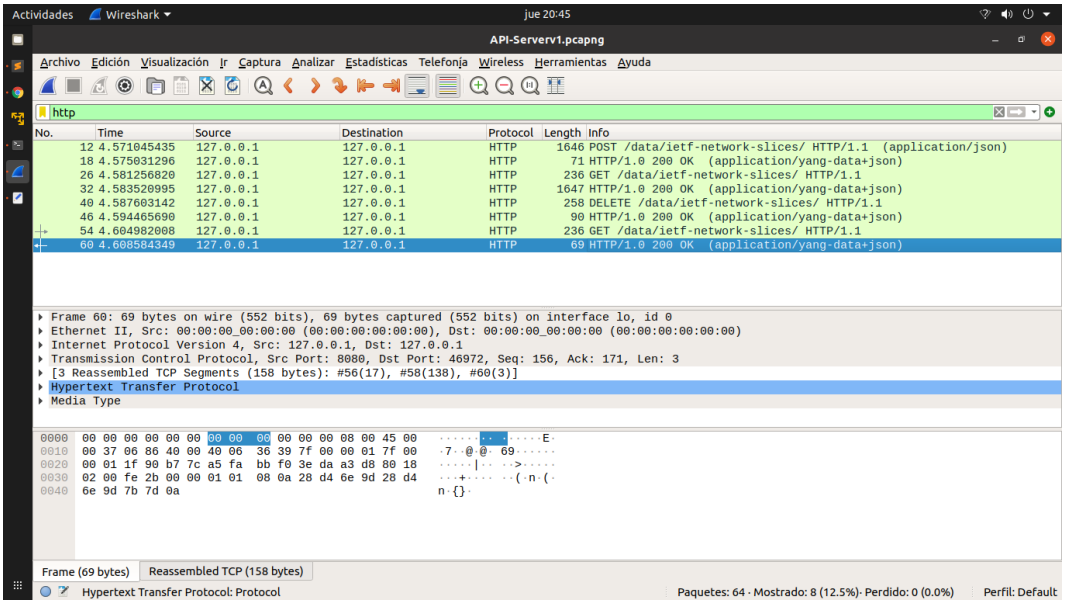

Figura 5.2: Figura sobre las trazas captadas por Whireshark

Como se puede observar en la figura 5.2, la primera petición que se le realiza al servidor es un POST, el cuál, en su cuerpo tiene la configuración de la red que se ha configurado desde el proceso y como se puede ver la petición que realizamos al servidor nos devuelve el código 200 **OK**. Este código se devuelve debido a que se ha guardado correctamente la configuración necesaria en el servidor o en la base de datos **Candidate**.

La segunda petición que se observa es GET, dónde nos devuelve la configuración de red que se ha enviado desde el proceso, si en el servidor se realiza correctamente la obtención de la configuración de la red. En la primera fase de desarrollo se obtiene del diccionario que hemos creado, en la segunda fase de desarrollo se obtiene de la base de datos **Candidate** y en las últimas fases de pruebas se obtiene de la base de datos **Running**.

La tercera petición que se observa en la traza es el DELETE y va acompañado de un GET para ver si la eliminación de la configuración de la red se ha realizado correctamente. En la primera fase del desarrollo se eliminaría del diccionario y en as siguientes fases de desarrollo se eliminaría de la base de datos **Candidate**.

A continuación, se mostrará la figura 5.3 y la figura 5.4 que muestran la salida por pantalla que realiza el proceso que se esta ejecutando en el ordenador local.

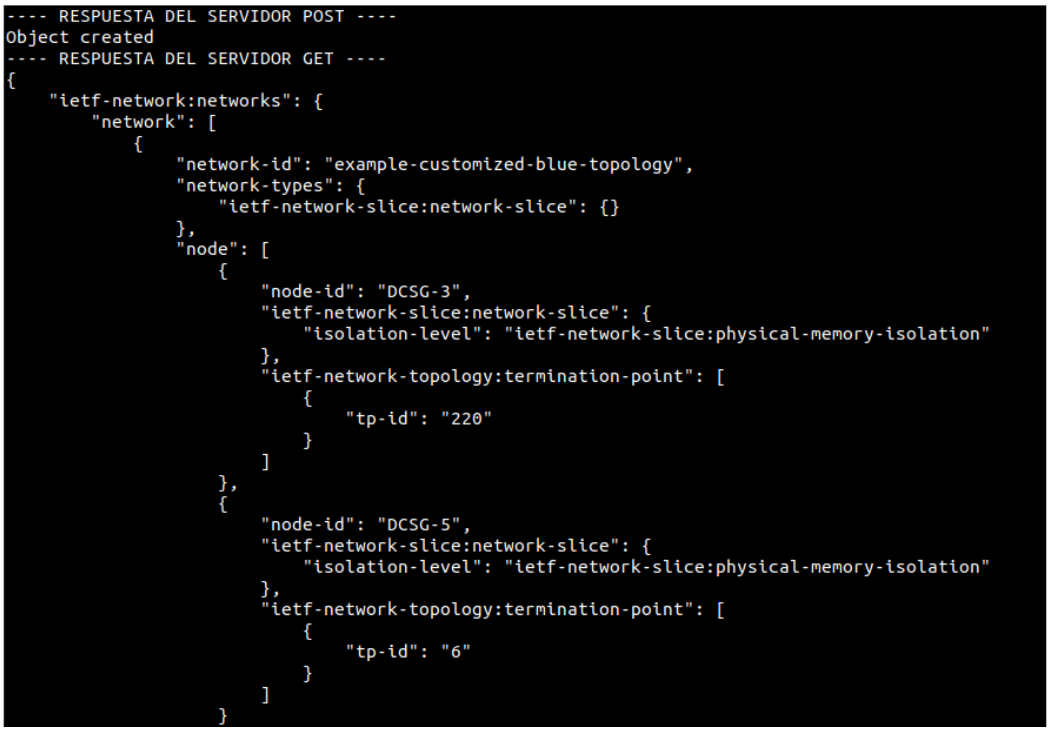

Figura 5.3: Figura de la primera parte de la salida por pantalla del proceso ejecutado por el servidor

Para probar toda la funcionalidad de los productores-consumidores con múltiples llamadas al servidor con diferentes configuraciones de red, se han creado dos hilos en el proceso principal. El primer hilo esta destinado a enviar peticiones POST al servidor para mandar las nuevas configuraciones que estamos creando en el proceso principal y, el segundo hilo esta destinado a realizar peticiones GET al servidor para obtener las configuraciones de red que han sido probadas. Este segundo hilo, durante las distintas fases de desarrollo ha obtenido de diferentes formas las configuraciones de red como se ha comentado anteriormente.

Por último, en el segundo productor-consumidor que es el que esta asociado al **NSO** se ha realizado un proceso con un hilo de escucha en la base de datos **Candidate** para que cuando haya una configuración de red disponible en el base de datos poder comprobarla y enviarla al **Volta Coordinator** y a **Running**.

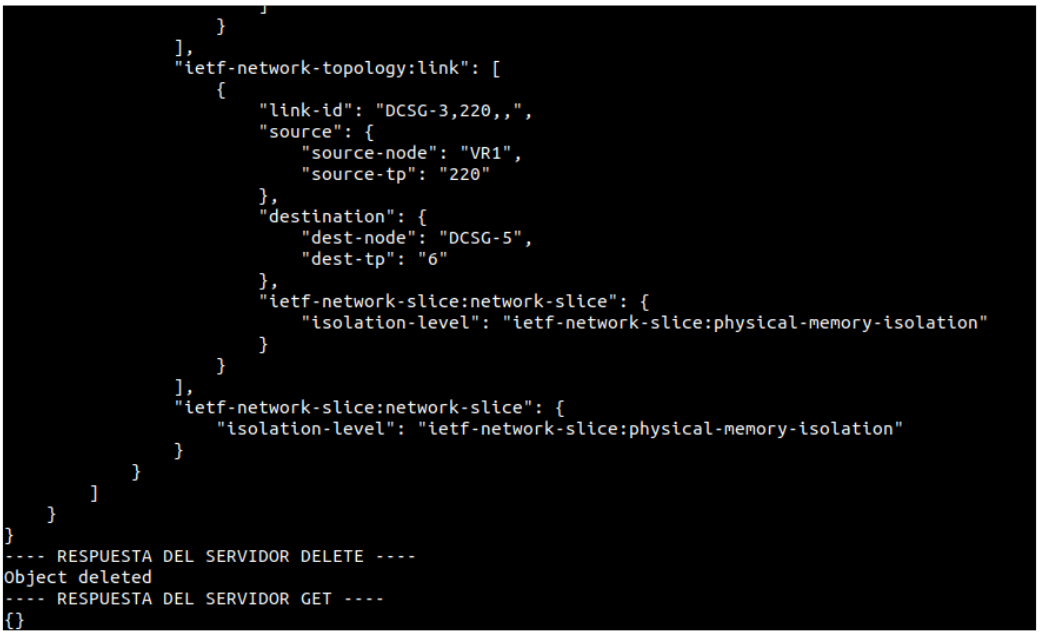

Figura 5.4: Figura de la primera parte de la salida por pantalla del proceso ejecutado por el servidor

6

Conclusiones y trabajo en un futuro

6.1. Conclusión

Las conclusiones que he llegado con este trabajo fin de grado es que la tecnología 5G va a llevar a toda la humanidad a dar un nuevo salto tecnológico, como se hizo con las anteriores generaciones. Esta generación de comunicaciones creo que es diferentes a las demás tecnologías debido a que se quiere mejorar mucho la comunicación entre todas las máquinas que estén conectadas a la red y, que el usuario de estas máquinas tenga la información que quieran con una latencia mínima y que al segundo o menos tiempo tenga lo que esta buscando. Esta generación también hace mejorar en los distintos campos de las diferentes industrias, debido a la alta velocidad de la comunicación y a la baja latencia, debido a esto se puede mejorar en el ámbito de los coches autónomos, de la realidad virtual o de la realidad aumentada. También se ha visto que con las slices que hay dentro de una red 5G va a mejorar la utilización de la red, debido a que la misma la red se va a poder utilizar para diferentes accesos y es válida para todo tipos de usuarios, debido a que una slice la pueda utilizar un usuario normal que este viendo una película en una plataforma de streaming y otra slice que este replicando nodos en la red la esté utilizando una empresa para desarrollar una plataforma de comunicaciones entre diferentes partes del mundo.

El proyecto se ha centrado en desarrollar un Network Slice Controller que permite realizar configuraciones de network slices utilizando soluciones abiertas que se están estandarizando en el IETF. Por un lado se ha utilizado una aplicación para solicitar slices dentro del módulo NSC-NBi, se ha realizado una arquitectura de base de datos siguiendo los mecanismos de Candidate y Running. Finalmente, se ha realizado un módulo Network Slice Orchestrator que ha permitido realizar la integración con la solución de Volta con sus equipos reales.

El trabajo realizado ha permitido conseguir una publicación en una conferencia internacional:

- **A. Alcalá**, S. Barguil, V. López, L. M. Contreras, C. Manso, P. Alemany, R. Casellas, R. Martínez, D. Gonzalez-Perez, X. Liu, J.M. Pulido, J.P. Fernandez-Palacios, R. Muñoz, R. Vilalta: *Multi-layer Transport Network Slicing with Hard and Soft Isolation*, in Proc. Optical Fiber Conference (OFC), Jun 2021.

6.2. Trabajo en un futuro

El proyecto desarrollado en este trabajo de fin de grado se puede mejorar de muchas formas, en primer lugar, para que sea más ameno la utilización de la aplicación creada para poder conectarse con el servidor se podría hacer una interfaz gráfica para que el usuario tenga un aspecto más visual de las creaciones de los nodos, enlaces de la red y sobretodo de la creación de la red acorde con sus necesidades. Debido a que es una tecnología muy cambiante puede darse cambios en los documentos de IETF escogidos son drafts y se tendrá que ir modificando la implementación con la evolución de los documentos, pero esto se podría hacer de manera automática de cuando salga un draft nuevo se podía descargar y crear de nuevo el servidor para poder seguir ejecutando la aplicación frente al nuevo NSC-NBi.

Otro trabajo en el futuro es adaptar nuestras llamadas a la aplicación con las diferentes necesidades que tienen nuestras aplicaciones de nuestra máquina. Se podía hacer un proceso que se este ejecutando en segundo plano y que haga una medición de los recursos que esta utilizando la aplicación sobre la red y pueda realizar una llamada al NSC-NBi para crear una nueva configuración red óptima para las aplicaciones que se están ejecutando en nuestra máquina.

Bibliografía

[1] P. Marsch, O. Bulakci, and M. Boldi, *5G system design*, ch. 1. 2019.

[2] A. Kaloxylos, C. Mannweiler, G. Zimmermann, M. D. Girolamo, P. Marsch, J. Belschner, R. T. O. B. A Tzanakaki, P. Spapis, P. Rost, P. Arnold, and N. Nikaein, *5G system design*, ch. 8. 2019.

[3] 5GPPP, "5gppp," 2021.

[4] 3GPP, "About 3gpp," 2021.

[5] J. M. Pulido, "Hard vs. soft network slicing: What's the difference (part 1)," 2020.

[6] J. M. Pulido, "Hard vs. soft network slicing: What's the difference (part 2)," 2020.

[7] J. M. Pulido, "Hard vs. soft network slicing: What's the difference (part 3)," 2020.

[8] tools.ietf, "draft-liu-teas-transport-network-slice-yang-02 - ietf network slice yang data model," 2021.

[9] M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Rfc 6241," 2011.

[10] S. Barguil, L. Contreras, V. Lopez, and O. G. de Dios, "Draft network slcies instantation," 2021.

[11] V. U. Manual, "Customer scripts documentation," 2020.

Apéndices

A

Multi-layer Transport Network Slicing with Hardand Soft Isolation

Multi-layer Transport Network Slicing with Hard and Soft Isolation

A. Alcalá[1], S. Barguil[1], V. López[2], L.M. Contreras[2], C. Manso[3], P. Alemany[3], R. Casellas[3], R. Martínez[3], D. González-Pérez[4], X. Liu[4], J.M. Pulido[4], J.P. Fernández-Palacios[2], R. Muñoz[3], R. Vilalta[3]

[1] *Universidad Autónoma de Madrid, Madrid, Spain*
[2] *Telefónica I+D, Madrid, Spain*
[3] *Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), Castelldefels (Barcelona), Spain*
[4] *Volta Networks, Barcelona, Spain*
e-mail: ricard.vilalta@cttc.es

Abstract: We validate the deployment of isolated transport network slices in IP over DWDM network. To this end, an isolated transport network slice is deployed using multi-layer isolation mechanisms based on OpenConfig and ONF Transport API.

1. Introduction

The provisioning of connectivity services that guarantee a specific set of Service Level Objectives (SLO) regarding network resources is expected to benefit many use cases, such as beyond 5G networks or NFV and data center interconnects. Transport network slices provide connectivity coupled with a set of specific network resources commitment between several endpoints over a shared network infrastructure [1].

Each slice is associated with a tenant. Each tenant can control and manage all its slices. As underlying resource multi-tenancy is supported, multiple isolation options are provided, such as soft slicing and hard slicing. Soft network slicing focuses on QoS mechanisms that provide a dynamic allocation of available network resources to different traffic classes. An example of a soft network slice is a VLAN assigned to a customer to carry voice/data traffic, usually transported by LxVPNs. On the other hand, hard network slicing provides component virtualization and replication. An example of a hard network slice is routers (physical or virtual) under the same administrative domain, where services are configured between routers.

The virtualization of the transport network can be exploited at the time of provisioning and orchestrating the virtualized service functions of the slice [2]. The idea leverages on the concept of Wide-area Infrastructure Manager (WIM) as defined by the ETSI NFV architecture as the element devoted to manage the virtualization capabilities in the WAN. Thus, it is assumed that a control entity will be in charge of handling the WAN transport connectivity for interconnecting given communication end-points. When referring to slices, such an entity could be associated to a Network Slice Controller (NSC), as defined in [1] either complementing or being part of an overarching SDN transport network control environment. A single transport network slice request can be decomposed into multiple control and management steps across all the network components involved. In order to provide the necessary transport network slice, NSC will determine the necessary resources allocation depending on the characteristics of the requested network slice (i.e., soft or hard). Once resources are allocated, they are configured on the network using multiple southbound interfaces (SBI).

an example interface between the NSC and the underlying optical controller is the Open Networking Foundation (ONF) Transport API (T-API) [3], which allows the underlying optical SDN controller topology export and connectivity service provisioning. ONF T-API allows the provisioning of connectivity services with specific connectivity constraints (including QoS requirements), which can later be mapped to specific network slice isolation levels.

Another well demonstrated data model is OpenConfig [4], which provides vendor-neutral YANG data models for various network elements, from routers to optical switches. The IP SDN Domain Controller will be responsible for allocating, instantiating and configuring the multiple (virtual) routers and interfaces depending on the required slice isolation level. OpenConfig data models usage is combined with several protocols such as gRPC or NETCONF. The gRPC protocols provide high performance and scalability to the proposed architecture.

This paper presents an end-to-end architecture to provide transport network slices deployed over multi-layer IP over DWDM networks. Several degrees of isolation (from hard to soft) might be required and implemented in the requested transport network slice. This is the first paper to explore transport network slice isolation using an IP over DWDM network. In order to validate this proposed architecture, we present a proof-of-concept in Telefonica and CTTC Laboratories.

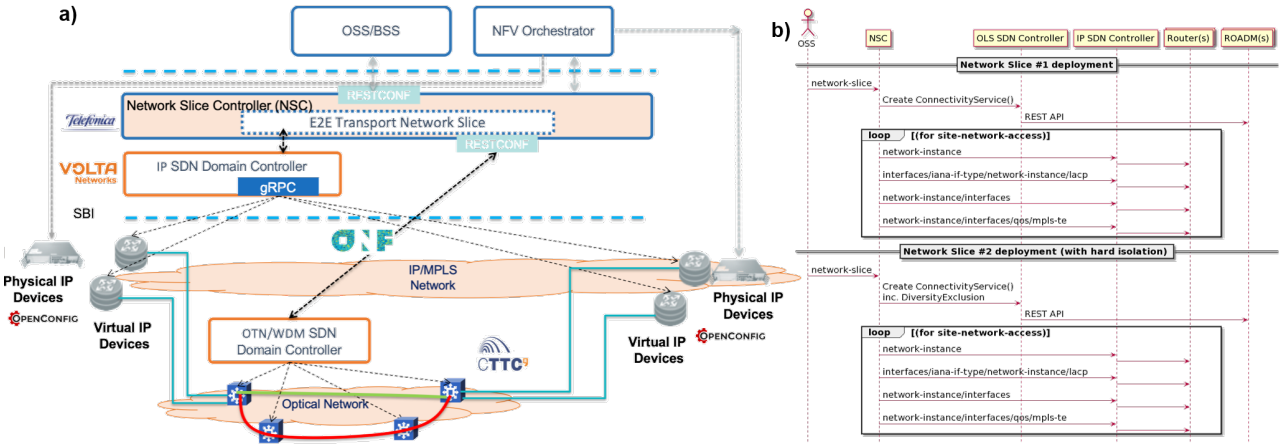

Fig. 1. a) Overall proposed architecture; and b) Sequence diagram for multiple network slice deployment with isolation characteristics.

2. Proposed architecture

Figure 1.a shows a network scheme of the approached concept. It consists of three main domains: SDN domain, IP domain, and optical domain.

The Network Slice Controller (NSC) realizes a transport network slice in the underlying transport infrastructure, maintains and monitors the state of its resources. The NSC will delegate on SDN Domain controllers [6] to configure the network resources. The NSC receives a transport network slice request from the Operation Support System and Business Support System (OSS/BSS). The request is modeled uing the YANG data model defined in [5] by means of the RESTCONF protocol [7]. Internal workflow for transport network slice life-cycle management is prepared on top of L2/L3 service management (SM) workflows. In order to interact with underlying IP and Optical Domain controllers via a RESTCONF client.

The OSS/BSS may request that the deployed network slice is isolated from any other network slices or different services delivered to the same customers. Naturally, other network slices or services must not negatively impact the requested transport network slice's delivery. There are several possibilities to provide this isolation, which can be provided at several degrees, such as dedicated allocation of resources for a specific slice or sharing some network resources. Figure 2.a shows the multiple isolation options that range from a hard slice to a soft slice: a) no-isolation, meaning that slices are not separated; b) physical-isolation, where slices are completely physically separated, for example, in different locations; c) logical-isolation, where slices are logically separated, only a certain degree of isolation is performed through QoS mechanisms; d) process-isolation, where slices include process and threads isolation; e) physical-network-isolation, where slices contain physically separated links; f) virtual-resource-isolation, where slices have dedicated virtual resources; g) network-functions-isolation, where Network Function (NF) are dedicated to a single network slice; h) service-isolation, where virtual resources and NFs are shared.

Figure 1.b shows the proposed workflow to deploy hard and soft transport network slices. In the workflow, two isolated network slices are deployed. The first one allocates a connectivity service to interconnect both IP layer domains (see for reference Figure 1.a). This triggers the necessary optical configuration mechanism to each of the underlying ROADMs (e.g., using OpenROADM protocol).

Once the connectivity service has been established, the NSC is responsible for requesting to IP SDN domain controller the necessary virtual routers (in the proposed scenario, two site-network-access are configured, one network instance on each site). Link Aggregation Control Protocol (LACP) is configured to each network instance. Then interfaces are aggregated and properly configured using dedicated VLAN or MPLS-TE mechanisms.

When a new isolated slice is requested, NSC can request a dedicated and isolated connectivity service to the underlying optical SDN controller. Figure 2.b details the available connectivity constraint to provide disjoint path selection using ONF Transport API. It consists of including a diversity exclusion constraint with the previous connectivity service identifier. Later, at IP layer novel virtual routers are deployed to provide the requested degree of isolation.

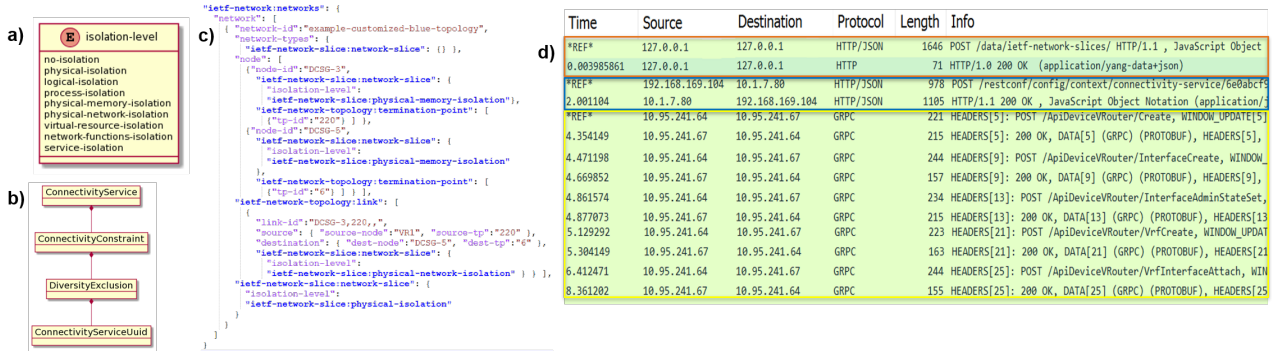

Fig. 2. a) Network Slice isolation levels; b) Disjoint path selection using ONF Transport API; c) Example of transport slice request; and d) Wireshark captures for deployment of a hard slice.

3. Results

The experimental testbed has been set-up at Telefónica (NSC deployment and Volta Elastic Virtual Routing Engine stacked in 7316 Edgecore hardware) and CTTC laboratories (including a T-API SDN controller over a flexi-grid 4-nodes DWDM network), as depicted in Figure 1.a. In order to properly set-up transport, we have introduced the transport network slice YANG data model in the RESTCONF server of NSC. An example of requested transport network slice is depicted in Figure 2.c. It can be observed that several levels of isolation can be requested per node, link and slice.

In order to validate the depicted sequence diagram in Figure 1.b, we provide the captured Wireshark traces among the complete system to deploy a complete hard slice request (the figure provides the selected significant traces). In this figure the different involved protocols can be observed. Firstly it shows the request for the transport network slice. It is quickly stored and answered, as it is processed after response (later an status update might be requested).

The transport network slice requests physical network isolation for the solicited link, thus a T-API connectivity service is requested including as connectivity constraints the diversity exclusion option. For that, we provide the identifiers from the previous established connectivity services and we obtain a disjoint path for the new requested slice, resulting in the requested physical network isolation (Figure 2.b). It can be appreciated that the connectivity service setup time is 2.001s. Consider that the ADRENALINE testbed already has the paths equalized.

Finally, the virtual routers (vRouter) are created and properly setup. The Wireshark capture shows the internal gRPC protocol from Volta Networks, which is equivalent to OpenConfig calls, for the vRouter deployment and configuration. It includes a call to create a new vRouter on top of Edgecore hardware. Then, interfaces are incorporated to the vRouter and configured. Later, Virtual routing and forwarding (VRF) table is setup and finally interfaces are attached to the VRF. This process set-up delay is of 8.36s.

4. Conclusions

We have presented and validated an end-to-end architecture that allows the deployment of transport network slices with several degrees of isolation. The results indicate the feasibility of deployment of multi-layer IP over DWDM transport network slices bsed on virtual roauters and optical disjoint paths to provide hard isolation. Soft isolation is instead reached through connectivity constraints and L2VPN service router configuration.

Acknowledgements

Work supported by the EC H2020 TeraFlow (101015857) and Spanish AURORAS (RTI2018-099178-I00) projects.

References

1. R. Rokui, et al., Definition of IETF Network Slices, IETF draft draft-ietf-teas-ietf-network-slice-definition-00, 2021.
2. R. Casellas, et al., Virtualization of disaggregated optical networks with open data models in support of network slicing. JOCN Feb 1;12(2):A144-54, 2020.
3. R. Vilalta, et al., Transport API extensions for the interconnection of multiple NFV infrastructure points of presence. OFC, 2019.
4. A. Shaikh, et al., Vendor-neutral network representations for transport SDN. OFC, 2016.
5. X. Liu, et al., IETF Network Slice YANG Data Model, IETF draft-liu-teas-transport-network-slice-yang-02, 2020.
6. L. Contreras, et al., FUSION: Standards-based SDN Architecture for Carrier Transport Network. CSCN, 2018.
7. A. Bierman, M. Bjorklund, K. Watsen, R. Fernando, RESTCONF protocol. IETF RFC 8040, 2017.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID